Ontological Queries: Rewriting and Optimization (Extended Version)*

Georg Gottlob[1,2], Giorgio Orsi[1,3], Andreas Pieris[1]

[1]*Department of Computer Science, University of Oxford, UK*
[2]*Oxford-Man Institute of Quantitative Finance, University of Oxford, UK*
[3]*Institute for the Future of Computing, University of Oxford, UK*

{georg.gottlob,giorgio.orsi,andreas.pieris}@cs.ox.ac.uk

Abstract

Ontological queries are evaluated against an ontology rather than directly on a database. The evaluation and optimization of such queries is an intriguing new problem for database research. In this paper we discuss two important aspects of this problem: query rewriting and query optimization. Query rewriting consists of the compilation of an ontological query into an equivalent query against the underlying relational database. The focus here is on soundness and completeness. We review previous results and present a new rewriting algorithm for rather general types of ontological constraints. In particular, we show how a conjunctive query against an ontology can be compiled into a union of conjunctive queries against the underlying database. Ontological query optimization, in this context, attempts to improve this process so to produce possibly small and cost-effective UCQ rewritings for an input query. We review existing optimization methods, and propose an effective new method that works for *linear Datalog*$^{\pm}$, a class of Datalog-based rules that encompasses well-known description logics of the *DL-Lite* family.

1 Introduction

This paper is about ontological query processing, an important new challenge to database research. We will review existing methods and propose new algorithms for compiling an *ontological query*, that is, a query against an ontology on top of a relational database, into a direct query against this database, and we will deal with optimization issues related to this process so as to obtain possibly small and efficient compiled queries. In this section, we first discuss a number of relevant concepts, and then illustrate query rewriting and optimization processes in the context of a small but non-trivial example.

Ontologies. The use of ontologies and ontological reasoning in companies, governmental organizations, and other enterprises has become widespread in recent years. An *ontology* is an *explicit specification of a conceptualization* of an area of interest [2], and consists of *a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts* [3]. To distinguish an enterprise ontology from a data dictionary, Dave McComb explicitly refers to the formal semantics of ontologies that enables automated processing and inferencing, while the interpretation of a data dictionary is strictly done by humans [4]. Moreover, ontologies have been adopted as high-level conceptual descriptions of the data contained in data repositories that are sometimes distributed and heterogeneous in the data models. Due to their high expressive power, ontologies are also substituting more traditional conceptual models such as UML class-diagrams and E/R schemata.

*This is an extended and revised version of the paper [1].

Description Logics. Description logics (DLs) are logical languages for expressing and modelling ontologies. The best known DLs are those underlying the *OWL* language[1]. The main ontological reasoning and query answering tasks in the complete OWL language, called *OWL Full*, are undecidable. For the most well-known decidable fragments of OWL, ontological reasoning and query answering is still computationally very hard, typically 2EXPTIME-complete.

In description logics, the ontological axioms are usually divided into two sets: The *ABox* (assertional box), which essentially contains factual knowledge such as "IBM is a company", denoted by $company(ibm)$, or "IBM is listed on the NASDAQ", which could be represented as a fact of the form $list\_comp(ibm, nasdaq)$, and a *TBox* (terminological box) which contains axioms and constraints that allow us, on the one hand, to infer new facts from those given in the ABox, and, on the other hand, to express restrictions such as keys. For example, a TBox may contain an axiom stating that for each fact $list\_comp(X, Y)$, Y must be a financial index, which in DL is expressed as $\exists list\_comp^- \sqsubseteq fin\_idx$. If the fact $fin\_idx(nasdaq)$ is not already present in the ABox, it can be derived via the above axiom from $list\_comp(ibm, nasdaq)$. Thus, the atomic query "$q(X) \leftarrow fin\_idx(X)$" would return *nasdaq* as one of the answers. Note that the axiom $\exists list\_comp^- \sqsubseteq fin\_idx$, which corresponds to an inclusion dependency, is *enforced by adding new tuples*, rather than just being *checked*. This is one main difference between ontological constraints and classical database dependencies. In database terms, the above axiom is to be interpreted more like a *trigger* than a classical constraint.

Ontology Based Data Access (OBDA). We are currently witnessing the marriage of ontological reasoning and database technology. In fact, this amalgamation consists in the realization of the obvious idea that ABoxes shall be implemented in form of a relational database, or even stored in classical RDBMSs. Moreover, very large existing databases are semantically enriched with ontological constraints. There are a number of recent commercial systems and experimental prototypes that extend RDBMSs with the possibility of querying an ontology that is rooted in a database (for examples, see Section 2). The main problem here is how to couple these two different types of technology smoothly and efficiently, and this is also the main theme of the present paper.

One severe obstacle to efficient OBDA is the already mentioned high computational complexity of query answering with description logics. The situation clearly worsens when the ABoxes of enterprise ontologies are very large databases. To tackle this problem, new, lightweight DLs have been designed, that guarantee *polynomial-time data complexity* for conjunctive query answering. This means that based on a fixed TBox, a fixed query can be answered in polynomial time over variable databases. The best-known and best-studied examples of such lightweight DLs are the *DL-Lite* [5] and $\mathcal{EL}$ (see, e.g., [6]) families. These languages can be considered tractable subclasses of OWL. It was convincingly argued that simple DLs such as DL-Lite or $\mathcal{EL}$ are sufficient for modelling an overwhelming number of applications.

More recently, the Datalog$^\pm$ family of description logics was introduced [7, 8, 9, 10]. Its syntax is based on classical first-order logic, more specifically, on variants of the well-known Datalog language [11]. The basic Datalog$^\pm$ rules are known as *tuple-generating dependencies* (TGDs) in the database literature [12]. Tractable DLs in this framework are *guarded Datalog*$^\pm$, which is noticeably more general than both DL-Lite and $\mathcal{EL}$, and the DLs *linear Datalog*$^\pm$ and *sticky-join Datalog*$^\pm$, which both encompass DL-Lite.

Besides being more expressive than DL-Lite, suitable Datalog$^\pm$ languages offer a more compact representation of the attributes of concepts and roles, since description logics are usually restricted to unary and binary predicates only. Consider, as an example, a relation $stock(\underline{\mathsf{id}}, \mathsf{name}, \mathsf{unit\text{-}price})$. Representing this relation in DL would require the introduction of a concept symbol *stock*, and of three attribute symbols *id*, *name* and *unit-price*. These entities must be then weaved together by the TBox formula $stock \sqsubseteq \exists id \sqcap \exists name \sqcap \exists unit\text{-}price$. Datalog$^\pm$ represents the relation in a natural way by means of a ternary predicate *stock*. In the same way, Datalog$^\pm$ provides a more natural syntax for many other DL formulae; for example, an inverse role assertion $r \sqsubseteq s^-$ is represented as a (full) TGD $r(X, Y) \rightarrow s(Y, X)$, while an existential restriction $p \sqsubseteq \exists r.q$ is represented as a (partial) TGD $p(X) \rightarrow \exists Y\, r(X, Y), q(Y)$.

[1] http://www.w3.org/TR/owl2-overview/

First-Order Rewritability. Polynomial-time tractability is often considered not to be good enough for efficient query processing. Ideally, one would like to achieve the same complexity as for processing SQL queries, or, equivalently, first-order (FO) queries. An ontology language $\mathcal{L}$ is *first-order rewritable* if, for every TBox Σ expressed in $\mathcal{L}$ and a query q, a first-order query q_Σ (called the *perfect rewriting*) can be constructed such that, given a database D, q_Σ evaluated over D yields exactly the same result as q evaluated against D and Σ. Since answering first-order queries is in the class AC_0 in data complexity [13], it immediately follows that under FO-rewritable TGDs, query answering is also in AC_0 in data complexity

This notion was first introduced by Calvanese et al. [5] in the concept of description logics. If a DL guarantees the FO-rewritability of each query under every TBox, we simply say that the logic is FO-rewritable. FO-rewritability is a most desirable property since it ensures that the reasoning process can be largely decoupled from data access. In fact, to answer query q, a separate software can compile q into q_Σ, and then just submit q_Σ as a standard SQL query to the DBMS holding D, where it is evaluated and optimized in the usual way.

Excitingly, it was shown that the members of the DL-Lite family, as well as the slightly more expressive language linear Datalog$^\pm$ are FO-rewritable. Moreover, even the much more expressive language of sticky-join Datalog$^\pm$ is FO-rewritable. For these languages, a pair $\langle \Sigma, q \rangle$, where q is a CQ, is rewritten as an SQL expression equivalent to a UCQ q_Σ. The research challenge we address in this paper is precisely the question of how to rewrite $\langle \Sigma, q \rangle$ to q_Σ correctly and efficiently. Let us illustrate this process by a small, but comprehensive example.

Consider the following relational schema $\mathcal{R}$ representing financial information about companies and their stocks:

$$\begin{array}{l} \mathit{stock}(\mathsf{id}, \mathsf{name}, \mathsf{unit\text{-}price}) \\ \mathit{company}(\mathsf{name}, \mathsf{country}, \mathsf{segment}) \\ \mathit{list\_comp}(\mathsf{stock}, \mathsf{list}) \\ \mathit{fin\_idx}(\mathsf{name}, \mathsf{type}, \mathsf{ref\text{-}mkt}) \\ \mathit{stock\_portf}(\mathsf{company}, \mathsf{stock}, \mathsf{qty}). \end{array}$$

The *stock* relation contains information about stocks such as the name, and the price per unit. The relation *company* contains information about companies; in particular, the name, the country, and the market segment of a company. The relation *list_comp* relates a stock to a financial index (i.e., NASDAQ, FTSE, NIKKEI) represented by the relation fin_idx which, in turn, contains information about the types of stocks in the index, and the reference market (e.g., London Stock Exchange). Finally, *stock_portf* relates companies to their stocks along with an indication of the amount of the investment.

Datalog$^\pm$ provides the necessary expressive power to extend $\mathcal{R}$ with ontological constraints in an easy and intuitive way. Examples of such constraints follow:

$$\begin{array}{lcl} \sigma_1 & : & \mathit{stock\_portf}(X, Y, Z) \rightarrow \exists V \exists W \ \mathit{company}(X, V, W) \\ \sigma_2 & : & \mathit{stock\_portf}(X, Y, Z) \rightarrow \exists V \exists W \ \mathit{stock}(Y, V, W) \\ \sigma_3 & : & \mathit{list\_comp}(X, Y) \rightarrow \exists Z \exists W \ \mathit{fin\_idx}(Y, Z, W) \\ \sigma_4 & : & \mathit{list\_comp}(X, Y) \rightarrow \exists Z \exists W \ \mathit{stock}(X, Z, W) \\ \sigma_5 & : & \mathit{stock\_portf}(X, Y, Z) \rightarrow \mathit{has\_stock}(Y, X) \\ \sigma_6 & : & \mathit{has\_stock}(X, Y) \rightarrow \exists Z \ \mathit{stock\_portf}(Y, X, Z) \\ \sigma_7 & : & \mathit{stock}(X, Y, Z) \rightarrow \exists V \exists W \ \mathit{stock\_portf}(V, X, W) \\ \sigma_8 & : & \mathit{stock}(X, Y, Z) \rightarrow \mathit{fin\_ins}(X) \\ \sigma_9 & : & \mathit{company}(X, Y, Z) \rightarrow \mathit{legal\_person}(X) \\ \delta_1 & : & \mathit{legal\_person}(X, Y, Z), \mathit{fin\_ins}(X, V, W) \rightarrow \bot. \end{array}$$

The first four TGDs set the "domain" and the "range" of the *stock_portf* and *list_comp* relations, respectively. TGDs σ_5 and σ_6 assert that *stock_portf* and *has_stock* are "inverse relations", while σ_7 expresses that each stock must belong to some stock portfolio. TGDs σ_8 and σ_9 model taxonomic relationships such as the facts that each stock is a financial instrument, and each company is a legal person. Finally, the negative constraint δ_1 (where $\bot$ denotes the truth constant *false*) states that legal persons and financial instruments are disjoint sets.

Figure 1: A (partial) rewriting for the Stock Exchange example.

$q^{[0]}(A,B,C) \leftarrow \mathit{fin\_ins}(A), \mathit{stock\_portf}(B,A,D), \mathit{company}(B,E,F), \mathit{list\_comp}(A,C), \mathit{fin\_idx}(C,G,H)$
$q^{[1]}(A,B,C) \leftarrow \mathit{fin\_ins}(A), \underline{\mathit{has\_stock}(A,B)}, \mathit{company}(B,E,F), \mathit{list\_comp}(A,C), \mathit{fin\_idx}(C,G,H)$
$q^{[2]}(A,B,C) \leftarrow \mathit{fin\_ins}(A), \mathit{has\_stock}(A,B), \underline{\mathit{stock\_portf}(B,E,F)}, \mathit{list\_comp}(A,C), \mathit{fin\_idx}(C,G,H)$
$q^{[3]}(A,B,C) \leftarrow \underline{\mathit{stock}(A,J,K)}, \mathit{has\_stock}(A,B), \mathit{stock\_portf}(B,E,F), \mathit{list\_comp}(A,C), \mathit{fin\_idx}(C,G,H)$
$\ldots$

Consider now the following conjunctive query q asking for all the triples $\langle a,b,c\rangle$, where a is a financial instrument owned by the company b and listed on c:

$$q(A,B,C) \leftarrow \mathit{fin\_ins}(A), \mathit{stock\_portf}(B,A,D), \mathit{company}(B,E,F), \mathit{list\_comp}(A,C), \mathit{fin\_idx}(C,G,H).$$

Since $\Sigma = \{\sigma_1, \ldots, \sigma_9\}$ is a set of linear TGDs, i.e., TGDs with single body-atom, query answering under Σ is FO-rewritable. Thus, it is possible to reformulate $\langle \Sigma, q\rangle$ to a first-order query q_Σ such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$. A naive rewriting procedure would use the TGDs of Σ as rewriting rules for the atoms in q to generate all the CQs of the perfect rewriting. Figure 1 shows a (partial) rewriting for q, where the query obtained at the i-th step is denoted as $q^{[i]}$, and the newly introduced atoms are underlined. In particular, $q^{[0]}$ is the given query q, $q^{[1]}$ is obtained from $q^{[0]}$ by using σ_6, $q^{[2]}$ is obtained from $q^{[1]}$ by applying σ_1, and $q^{[3]}$ is obtained from $q^{[2]}$ by using σ_8.

The complete perfect rewriting contains more than 200 queries executing more than 1000 joins. However, by exploiting the set of constraints, it is possible to eliminate redundant atoms in the generated queries, and thus reduce the number of the CQs in the rewritten query. For example, in the given query q above it is possible to eliminate the atom $\mathit{fin\_ins}(A)$ since, due to the existence of the TGDs σ_2 and σ_8 in Σ, if the atom $\mathit{stock\_portf}(B,A,D)$ is satisfied, then immediately the atom $\mathit{fin\_ins}(A)$ is also satisfied. Notice that by eliminating a redundant atom from a query, we also eliminate all the atoms that are generated starting from it during the rewriting process. Moreover, due to the TGD σ_3, if the atom $\mathit{list\_comp}(A,C)$ in q is satisfied, then the atom $\mathit{fin\_idx}(C,G,H)$ is also satisfied, and therefore can be eliminated. Finally, due to the TGD σ_1, if the atom $\mathit{stock\_portf}(B,A,D)$ is satisfied, then the atom $\mathit{company}(B,E,F)$ is also satisfied, and hence is redundant. The query that has to be considered as input of the rewriting process is therefore $q(A,B,C) \leftarrow \mathit{stock\_portf}(B,A,D), \mathit{list\_comp}(A,C)$ that produces a perfect rewriting containing the following two queries executing only two joins:

$$q(A,B,C) \leftarrow \mathit{list\_comp}(A,C), \mathit{stock\_portf}(B,A,D)$$
$$q(A,B,C) \leftarrow \mathit{list\_comp}(A,C), \mathit{has\_stock}(A,B).$$

Contributions and Roadmap. After a review of previous work on ontology based data access in the next section, and some formal definitions and preliminaries in Section 3, we present a short overview of the Datalog$^\pm$ family in Section 4. We then proceed with new research results. In Section 5, we propose a new rewriting algorithm that improves the one stated in [14] by substantially reducing the number of redundant queries in the perfect rewriting. In Section 6, we present a polynomial-time optimization strategy based on the early-pruning of redundant atoms produced during the rewriting process. An implementation and experimental evaluation of the new method is discussed in Section 7. We also discuss the relationship between our optimization technique and optimal query minimization algorithms such as the *chase & back-chase* algorithm [15]. We conclude with a brief outlook on further research.

2 Ontology Based Data Access

Answering queries under constraints and the related optimization techniques are important topics in data management beyond the obvious research interest. In particular, they are profitable opportunities for companies that need to deliver efficient and effective data management solutions to

their customers. This trend is becoming even more evident as a plethora of robust systems and APIs for Semantic Web data management proposed in the recent years. These systems span from open-source solutions such as Virtuoso[2], Sesame[3], RDFSuite [16], KAON[4] and Jena[5], to commercial implementations such as the semantic extensions implemented in Oracle Database 11g R2 [17] and BigOWLLim[6]. In this Section we briefly analyze the systems providing rewriting-based access to databases under ontological constraints, and we highlight some crucial points that we want to address in this work.

We first present the class of constraints identified by the members of the DL-Lite family [5], namely, DL-Lite$_{\mathcal{A}}$, DL-Lite$_{\mathcal{F}}$, and DL-Lite$_{\mathcal{R}}$, underlying the W3C OWL-QL profile of the OWL language. These constraints correspond to unary and binary *inclusion dependencies* combined with a restricted form of *key constraints*. In order to perform query answering under this class of constraints, a rewriting algorithm, introduced in [5] and implemented in the QuOnto system, reformulates the given query into unions of conjunctive queries. The size of the reformulated query is unnecessarily large due to a number of reasons. In the first place, *(i)* basic optimization techniques such as the identification of the connected components in the body of the input query, or the computation of any form of query decomposition [18], are not applied. Moreover, *(ii)* the fact that the given set of constraints can be used to identify existential joins in the reformulated query which can be eliminated is not exploited. Finally, *(iii)* the factorization step (which is needed to guarantee completeness) is applied exhaustively, and as a result many superfluous queries are generated.

Peréz-Urbina et al. [19] proposed an alternative resolution-based rewriting algorithm, implemented in the Requiem system, that addressed the issue of the useless factorizations (and therefore of the redundant queries generated due to this weakness) by directly handling existential quantification through proper functional terms. The algorithm has then been extended to more expressive DL languages [19]. In this case the output of the rewriting is a Datalog program.

Rosati et al. [20] recently proposed a very sophisticated rewriting technique, implemented in the Presto system, that addresses some of the issues described above. In particular, *(i)* the unnecessary existential joins are eliminated by resorting to the concept of *most-general subsumees*, which also avoids the unnecessary factorizations, and *(ii)* the connectivity of the given query is checked before executing the algorithm; in case the query is not connected, Presto splits the query in connected components and rewrites them separately. Notice that Presto produces a non-recursive Datalog program, and not a union of conjunctive queries. This allows the "hiding" of the exponential blow-up inside the rules instead of generating explicitly the disjunctive normal form. The final rewriting is exponential only in the number of non-eliminable existential joins, but not in the size of the input query.

The approaches presented above have been proven very effective when applied to very particular classes of description logic constraints. Following a more general approach for ontological query answering, Calì et al. [14] presented a backward-chaining rewriting algorithm which is able to deal with arbitrary sets of TGDs, providing that the class of TGDs under consideration satisfies suitable syntactic restrictions that guarantee the termination of the algorithm. However, this algorithm is inspired by the original QuOnto algorithm and inherits all its drawbacks.

Despite the possibly exponential number of queries to be constructed, we know that all such queries are independent from each other, and thus they can be easily executed in parallel threads and distributed on multiple processors. Notice that a non-recursive Datalog program is not equally easy to distribute. Moreover, the optimizations implemented in current DBMS systems for (unions of) conjunctive queries are much more advanced than those implemented for the positive existential first-order queries resulting from the translation of a non-recursive Datalog program into a concrete query language such as SQL. It is clear that a trade-off between these two approaches must be found

[2]http://virtuoso.openlinksw.com/
[3]http://www.openrdf.org/
[4]http://kaon.semanticweb.org/
[5]http://jena.sourceforge.net/
[6]http://www.ontotext.com/owlim/

in order to exploit as much as possible the current optimization techniques, while keeping the size of the rewriting reasonably small in order to make the execution of it feasible in practice.

A related research field is that of query minimization [21], in particular, in presence of views and constraints [22, 15]. Given a conjunctive query q, and a set of constraints Σ, the goal is to find all the minimal equivalent reformulations of q under the constraints of Σ. The most interesting approach in this respect is the chase & back-chase algorithm (C&B) [15], implemented in the MARS system [23]. The algorithm freezes the atoms of $body(q)$ and, by considering them as a database D_q, applies the following two steps. During the *chase-step*, the chase of D_q w.r.t. Σ is constructed, and then the atoms of $chase(D_q, \Sigma)$ are considered as the body-atoms of a query q_u, called the *universal plan*. The *back-chase step* considers all the possible subsets of the atoms of $body(q_u)$, starting from those with a single-atom, which are then considered as the body of a query q'. Whenever there exists a containment mapping from $body(q_u)$ to $chase(D_{q'}, \Sigma)$, where $D_{q'}$ is the database obtained by freezing $body(q')$, then q' is an equivalent reformulation of q. Moreover, every time an equivalent reformulation q' is found, the back-chase does not consider any of the supersets of the atoms of $body(q')$ because they will be automatically implied by the atoms of q', and therefore the produced query would be redundant. This particular exploration strategy guarantees the minimality of the reformulations. A non-negligible drawback of this approach is the fact that we need to compute the chase of D_q w.r.t. Σ, and also the chase for the (exponentially many) databases $D_{q'}$ w.r.t Σ. Clearly, this makes the procedure computationally expensive.

3 Preliminaries

In this section we recall some basics on relational databases, conjunctive queries, tuple-generating dependencies, and the chase procedure.

3.1 Relational Databases and Conjunctive Queries

Consider two pairwise disjoint (infinite) sets of symbols Δ_c and Δ_z such that: Δ_c is a set of *constants* (which constitutes the domain of a database), and Δ_z is a set of *labeled nulls* (used as placeholders for unknown values). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. Throughout the paper, we denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$, where $k \geq 0$, and by $[n]$ the set $\{1, \ldots, n\}$, for any $n \geq 1$.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicate symbols*), each with its associated arity. A *position* $r[i]$ (or $\langle r, i \rangle$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument. A *term* t is a constant, labeled null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where $r \in \mathcal{R}$ has arity n, and $t_1, \ldots, t_n$ are terms. Conjunctions of atoms are often identified with the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \to S_2$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution h from the set of terms of A_1 to the set of terms of A_2 such that: *(i)* if $t \in \Delta_c$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2. The notion of homomorphism naturally extends to conjunctions of atoms.

A *relational instance* (or simply *instance*) I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where $r \in \mathcal{R}$ has arity n and $\mathbf{t} \in (\Delta_c \cup \Delta_z)^n$. A *database* is a finite relational instance. A *conjunctive query* (CQ) q of arity n over a schema $\mathcal{R}$ is a formula of the form $q(\mathbf{X}) \leftarrow \phi(\mathbf{X}, \mathbf{Y})$, where $\phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, and q is an n-ary predicate. $\phi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $body(q)$, and $q(\mathbf{X})$ is the head of q, denoted as $head(q)$. A *Boolean conjunctive query* (BCQ) is a CQ of arity zero. The *answer* to a CQ q of arity n over an instance I, denoted as $q(I)$, is the set of all n-tuples $\mathbf{t} \in (\Delta_c)^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Delta_c \cup \Delta_z$ such that $h(\phi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over I, denoted as $I \models q$, iff $\langle\rangle \in q(I)$. A *union of CQs* (UCQ) Q of arity n is a set of CQs, where each $q \in Q$ has

the same arity n and uses the same predicate symbol in the head. The answer to Q over an instance I, denoted as $Q(I)$, is defined as the set of tuples $\{\mathbf{t} \mid \text{there exists } q \in Q \text{ such that } \mathbf{t} \in q(I)\}$.

3.2 Tuple-Generating Dependencies

A *tuple-generating dependency* (TGD) σ over a schema $\mathcal{R}$ is a first-order formula $\forall\mathbf{X}\forall\mathbf{Y}\,\phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\mathbf{Z}\,\psi(\mathbf{X},\mathbf{Z})$, where $\phi(\mathbf{X},\mathbf{Y})$ and $\psi(\mathbf{X},\mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $body(\sigma)$ and $head(\sigma)$, respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. Such σ is satisfied by an instance I for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\phi(\mathbf{X},\mathbf{Y})) \subseteq I$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X},\mathbf{Z})) \subseteq I$.

We now define the notion of *query answering* under TGDs. Given a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D,\Sigma)$, is the set of all instances I such that $I \models D \cup \Sigma$, which means that $I \supseteq D$ and I satisfies Σ. The *answer* to a CQ q w.r.t. D and Σ, denoted as $ans(q,D,\Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(I) \text{ for each } I \in mods(D,\Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, iff $ans(q,D,\Sigma) \neq \varnothing$. Note that query answering under general TGDs is undecidable [24], even when the schema and the set of TGDs are fixed [25]. We recall that the two problems of answering CQs and BCQs under TGDs are equivalent [21, 26]. Roughly speaking, we can enumerate the polynomially many tuples of constants which are possible answers to q, and then, instead of answering the given query q, we answer the polynomially many BCQs that we obtain by replacing the variables in the body of q with the appropriate constants. A certain tuple $\mathbf{t}$ of constants is in the answer of q iff the answer to the BCQ that we obtain from $\mathbf{t}$ is positive. Henceforth, we thus focus only on the BCQ answering problem.

3.3 The TGD Chase

The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [27], and later for checking query containment [28]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*.

TGD Chase Rule: Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma : \phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists\mathbf{Z}\,\psi(\mathbf{X},\mathbf{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\phi(\mathbf{X},\mathbf{Y})) \subseteq D$ then: *(i)* define $h' \supseteq h$ such that $h'(Z_i) = z_i$, for each $Z_i \in \mathbf{Z}$, where $z_i \in \Delta_z$ is a "fresh" labeled null not introduced before, and *(ii)* add to D the set of atoms in $h'(\psi(\mathbf{X},\mathbf{Z}))$, if not already in D.

Given a database D and a set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase rule in a breadth-first fashion, which leads as result to a (possibly infinite) chase for D and Σ, denoted as $chase(D,\Sigma)$. For the formal definition of the chase algorithm we refer the reader to [8].

The (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each instance $I \in mods(D,\Sigma)$, there exists a homomorphism from $chase(D,\Sigma)$ to I [26, 29]. Using this fact it can be shown that $D \cup \Sigma \models q$ iff $chase(D,\Sigma) \models q$, for every BCQ q.

4 The Datalog$^{\pm}$ Family

In this section we present the main Datalog$^{\pm}$ languages under which query answering is decidable, and (almost in all cases) also tractable in data complexity.

4.1 Decidability Paradigms

We first discuss the three main paradigms for ensuring decidability of query answering, namely, chase termination, guardedness and stickiness.

Chase Termination. In this case the chase always terminates and produces a finite universal model U. Thus, given a query we just need to evaluate it over the finite database U. The most notable syntactic restriction of TGDs guaranteeing chase termination is *weak-acyclicity*, which is defined by means of a graph-based condition, for which we refer the reader to the landmark paper [29]. Roughly speaking, in the chase constructed under a weakly-acyclic set of TGDs over a schema $\mathcal{R}$, only a finite number of distinct values can appear at any position of $\mathcal{R}$, and thus after finitely many steps the chase procedure terminates. It is known that query answering under a weakly-acyclic set of TGDs is PTIME-complete [29] and 2EXPTIME-complete [10] in data and combined complexity, respectively. More general syntactic restrictions that guarantee chase termination were proposed in [26] and [30].

Guardedness. *Guarded* TGDs, introduced and studied in [25], have an atom in their body, called the *guard*, that contains all the universally quantified variables. For example, the TGD $r(X,Y), s(X,Y,Z) \rightarrow \exists W s(Z,X,W)$ is guarded via the guard atom $s(X,Y,Z)$, while the TGD $r(X,Y), r(Y,Z) \rightarrow r(X,Z)$ is not. Decidability of query answering follows from the fact that the chase constructed under a set of guarded TGDs has the bounded treewidth property, i.e., is a "tree-like" structure. The data and combined complexity of query answering under a set of guarded TGDs is PTIME-complete [7] and 2EXPTIME-complete [25], respectively.

Linear TGDs, proposed in [7], is a FO-rewritable variant of guarded TGDs. A TGD is linear iff it contains only one atom in its body. Obviously a linear TGD is trivially guarded since the singleton body-atom is automatically a guard. Linear TGDs are more expressive than the well-known class of inclusion dependencies. Query answering under linear TGDs is in the highly tractable class AC_0 in data complexity [7]. The same problem is PSPACE-complete in combined complexity; this result is immediately implied by results in [28].

An expressive class, which forms a generalization of guarded TGDs, is the class of *weakly-guarded* sets of TGDs introduced in [25]. Intuitively speaking, a set Σ of TGDs is weakly-guarded iff in the body of each TGD of Σ there exists an atom, called the *weak-guard*, that contains all the universally quantified variables that appear only at positions where a "fresh" null of Δ_z can appear during the construction of the chase. Query answering under a weakly-guarded set of TGDs is EXPTIME-complete [25] and 2EXPTIME-complete [25] in data and combined complexity, respectively.

Stickiness. In this paragraph we present a Datalog$^{\pm}$ language (and its extensions), which hinges on a paradigm that is very different from guardedness. *Sticky* sets of TGDs are defined formally by an efficiently testable condition involving variable-marking [9]. In what follows we just give an intuitive definition of this class. For every database D, assume that during the construction of chase of D under a sticky set of TGDs, we apply a TGD $\sigma \in \Sigma$ that has a variable V appearing more than once in its body; assume also that V maps (via homomorphism) on the symbol z, and that by virtue of this application the atom $\underline{a}$ is introduced. In this case, for each atom $\underline{b}$ in $body(\sigma)$, we say that $\underline{a}$ is *derived* from $\underline{b}$. Then, we have that z appears in $\underline{a}$ and in all atoms resulting from some chase derivation sequence starting from $\underline{a}$, "sticking" to them (hence the name "sticky" sets of TGDs). Interestingly, sticky sets of TGDs are FO-rewritable, and thus query answering is feasible in AC_0 in data complexity [9]. Combined complexity of query answering is known to be EXPTIME-complete [9].

In [10] the FO-rewritable class of *sticky-join* sets of TGDs, that captures both linear TGDs and sticky sets of TGDs, is introduced. Similarly to sticky sets of TGDs, sticky-join sets are defined formally by a testable condition based on variable-marking. However, this variable-marking procedure is more sophisticated than the one used for sticky sets, and due to this fact the problem of identifying sticky-join sets of TGDs is harder than the one of identifying sticky sets. In particular, given a set Σ of TGDs, we can decide in PTIME whether Σ is sticky, while the problem whether Σ is sticky-join is PSPACE-complete. Note that the data and combined complexity of query answering under sticky and sticky-join sets of TGDs coincide.

4.2 Additional Features

In this subsection we briefly discuss how the languages presented above can be combined with negative constraints and key dependencies, without altering the complexity of query answering.

Negative Constraints. A *negative constraint* (NC) ν over a schema $\mathcal{R}$ is a first-order formula $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\bot$ denotes the truth constant *false*. NCs are vital when representing ontologies (see, e.g., [7, 9]), as well as conceptual schemas such as Entity-Relationship diagrams (see, e.g., [31, 32]). With NCs we can assert, for example, that students and professors are disjoint sets: $\forall X\, \mathit{student}(X), \mathit{professor}(X) \rightarrow \bot$. Also, we can state that a student cannot be the leader of a research group: $\forall X \forall Y\, \mathit{student}(X), \mathit{leads}(X, Y) \rightarrow \bot$.

It is known that checking NCs is tantamount to query answering [7]. In particular, given an instance I, a set $\Sigma_\bot$ of NCs, and a set Σ of TGDs, for each NC ν of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, we answer the BCQ $q_\nu() \leftarrow \phi(\mathbf{X})$. If at least one of such queries answers positively, then $I \cup \Sigma \cup \Sigma_\bot \models \bot$ (i.e., the theory is inconsistent), and therefore $I \cup \Sigma \cup \Sigma_\bot \models q$, for every BCQ q; otherwise, given a BCQ q, we have $I \cup \Sigma \cup \Sigma_\bot \models q$ iff $I \cup \Sigma \models q$, i.e., we can answer q by ignoring the set of NCs.

Key Dependencies. It is well-known that the interaction of general TGDs and key dependencies (KDs) leads to undecidability of query answering [33]; we assume that the reader is familiar with the notion of KD (see, e.g., [34]). Thus, the classes of TGDs presented above cannot be combined arbitrarily with KDs. Suitable syntactic restrictions are needed in order to ensure decidability of query answering.

A crucial concept towards this direction is separability [35], which formulates a controlled interaction of TGDs and KDs. Formally speaking, a set $\Sigma = \Sigma_T \cup \Sigma_K$ over a schema $\mathcal{R}$, where Σ_T and Σ_K are sets of TGDs and KDs, respectively, is *separable* iff for every instance I for $\mathcal{R}$, either I violates Σ_K, or for every BCQ q over $\mathcal{R}$, $I \cup \Sigma \models q$ iff $I \cup \Sigma_T \models q$. Notice that separability is a semantic notion. A sufficient syntactic criterion for separability of TGDs and KDs is given in [7]; TGDs and KDs satisfying the criterion are called *non-conflicting*.

Obviously, in case of non-conflicting sets of TGDs and KDs, we just need to perform a preliminary check whether the given instance satisfies the KDs, and if this is the case, then we eliminate them, and proceed by considering only the set of TGDs. This preliminary check can be done using negative constraints. For example, to check whether the KD $\mathit{key}(r) = \{1\}$, stating that the first attribute of the binary relation r is a key attribute, is satisfied by the database D, we just need to check whether the database $D_{\neq}$ obtained by adding to D the set of atoms $\{\mathit{neq}(a, b) \mid a \neq b, \text{ and } a, b \text{ are constants occurring in } D\}$, where neq is an auxiliary predicate, satisfies the negative constraint $r(X, Y), r(X, Z), \mathit{neq}(Y, Z) \rightarrow \bot$. The atom $\mathit{neq}(a, b)$ implies that a and b are different constants. Since, as already mentioned, checking NCs is tantamount to query answering, we immediately get that the complexity of query answering under non-conflicting sets of TGDs and KDs is the same as in the case of TGDs only.

Interestingly, by combining non-conflicting linear (or sticky) sets of TGDs and KDs with NCs, we get strictly more expressive formalisms than the most widely-adopted tractable ontology languages, in particular DL-Lite$_\mathcal{A}$, DL-Lite$_\mathcal{F}$ and DL-Lite$_\mathcal{R}$, without loosing FO-rewritability, and consequently high tractability of query answering in data complexity. For more details, we refer the interested reader to [7, 9].

5 Datalog$^\pm$ for OBDA

In this section we consider the problem of BCQ answering under the FO-rewritable members of the Datalog$^\pm$ family, namely, linear, sticky and sticky-join sets of TGDs. Given a BCQ q and a set Σ of TGDs, the actual computation of the rewriting is done by applying a backward-chaining resolution procedure using the rules of Σ as rewriting rules. Our algorithm optimizes the algorithm presented in [14] by greatly reducing the number of BCQs in the rewriting, and therefore improves the overall performance of query answering. Before going into the details of the rewriting algorithm, we first give some useful notions.

A set of atoms $A = \{\underline{a}_1, \ldots, \underline{a}_n\}$, where $n \geqslant 2$, *unifies* if there exists a substitution γ, called *unifier* for A, such that $\gamma(\underline{a}_1) = \ldots = \gamma(\underline{a}_n)$. A *most general unifier (MGU)* for A is a unifier for A, denoted as γ_A, such that for each other unifier γ for A, there exists a substitution γ' such that $\gamma = \gamma' \circ \gamma_A$. Notice that if a set of atoms unify, then there exists a MGU. Furthermore, the MGU for a set of atoms is unique (modulo variable renaming). The MGU for a singleton set $\{\underline{a}\}$ is defined as the identity substitution on the set of terms that occur in $\underline{a}$.

Let us now give some auxiliary results which will allow us to simplify our later technical definitions and proofs. The first such lemma states that we can restrict our attention on TGDs that have only one head-atom.

Lemma 1 *BCQ answering under (general) TGDs and BCQ answering under TGDs with just one head-atom are* LOGSPACE*-equivalent problems.*

Proof. It suffices to show that BCQ answering under (general) TGDs can be reduced in LOGSPACE to BCQ answering under TGDs with just one head-atom. Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. We construct Σ' from Σ by applying the following procedure. For each TGD $\sigma \in \Sigma$, where $\mathit{head}(\sigma) = \{\underline{a}_1, \ldots, \underline{a}_k\}$ and $\mathbf{X}$ is the set of variables that occur in $\mathit{head}(\sigma)$, replace σ with the following set of TGDs:

$$\begin{array}{rcl} \mathit{body}(\sigma) & \rightarrow & r_\sigma(\mathbf{X}), \\ r_\sigma(\mathbf{X}) & \rightarrow & \underline{a}_1, \\ r_\sigma(\mathbf{X}) & \rightarrow & \underline{a}_2, \\ & \vdots & \\ r_\sigma(\mathbf{X}) & \rightarrow & \underline{a}_k, \end{array}$$

where r_σ is an auxiliary predicate not occurring in $\mathcal{R}$ having the same arity as the number of variables in $\mathbf{X}$. It is not difficult to see that the above construction is feasible in LOGSPACE. By construction, except for the atoms with an auxiliary predicate, $\mathit{chase}(D, \Sigma)$ and $\mathit{chase}(D, \Sigma')$ coincide. The auxiliary predicates, being introduced only during the above transformation, do not match any predicate symbol in q, and hence $\mathit{chase}(D, \Sigma) \models q$ iff $\mathit{chase}(D, \Sigma') \models q$, or, equivalently, $D \cup \Sigma \models q$ iff $D \cup \Sigma' \models q'$. □

The next lemma implies that we can restrict our attention on TGDs that have only one existentially quantified variable which occurs only once.

Lemma 2 *BCQ answering under (general) TGDs and BCQ answering under TGDs with at most one existentially quantified variable that occurs only once are* LOGSPACE*-equivalent problems.*

Proof. It suffices to show that BCQ answering under (general) TGDs can be reduced in LOGSPACE to BCQ answering under TGDs that have at most one existentially quantified variable which occurs only once. Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. We construct Σ' from Σ by applying the following procedure. For each TGD $\sigma \in \Sigma$, where $\{X_1, \ldots, X_n\}$, for $n \geqslant 1$, is the set of variables that occur both in $\mathit{body}(\sigma)$ and $\mathit{head}(\sigma)$, and $\{Z_1, \ldots, Z_m\}$, for $m > 1$, is the set of the existentially quantified variables of σ, replace σ with the following set of TGDs:

$$\begin{array}{rcl} \mathit{body}(\sigma) & \rightarrow & \exists Z_1\, r_\sigma^1(X_1, \ldots, X_n, Z_1), \\ r_\sigma^1(X_1, \ldots, X_n, Z_1) & \rightarrow & \exists Z_2\, r_\sigma^2(X_1, \ldots, X_n, Z_1, Z_2), \\ & \vdots & \\ r_\sigma^{m-1}(X_1, \ldots, X_n, Z_1, \ldots, Z_{m-1}) & \rightarrow & \exists Z_m\, r_\sigma^m(X_1, \ldots, X_n, Z_1, \ldots, Z_m), \\ r_\sigma^m(X_1, \ldots, X_n, Z_1, \ldots, Z_m) & \rightarrow & \mathit{head}(\sigma), \end{array}$$

where r_σ^i is an auxiliary predicate of arity $n + i$, for each $i \in [m]$. It is easy to see that the above procedure can be carried out in LOGSPACE. By construction, except for the atoms with an auxiliary predicate, $\mathit{chase}(D, \Sigma)$ and $\mathit{chase}(D, \Sigma')$ are the same (modulo bijective variable renaming).

The auxiliary predicates, being introduced only during the above construction, do not match any predicate symbol in q, and hence $chase(D, \Sigma) \models q$ iff $chase(D, \Sigma') \models q$, or, equivalently, $D \cup \Sigma \models q$ iff $D \cup \Sigma' \models q$. □

Since the transformations given above preserve the syntactic condition of linear, sticky and sticky-join sets of TGDs, henceforth we assume w.l.o.g. that every TGD has just one atom in its head which contains only one existentially quantified variable that occurs only once. In the rest of the paper, for notational convenience, given a TGD σ, we denote by π_σ the position in $head(\sigma)$ at which the existentially quantified variable occurs.

We now give the notion of *applicability* of a TGD to a set of body-atoms of a query. Let us assume w.l.o.g that the variables that appear in the query, and those that appear in the TGD, constitute two disjoint sets. Given a BCQ q, a variable is called *shared* in q if it occurs more than once in $body(q)$. Notice that in the case of (non-Boolean) CQs, a variable is shared in q if it occurs more than once in q (considering also the head of q and not just its body).

Definition 1 (Applicability) *Consider a BCQ q over a schema $\mathcal{R}$, and a TGD σ over $\mathcal{R}$. Given a set of atoms $A \subseteq body(q)$ that unifies, we say that σ is* applicable *to A if the following conditions are satisfied: (i) the set $A \cup \{head(\sigma)\}$ unifies, and (ii) for each $\underline{a} \in A$, if the term at position π in $\underline{a}$ is either a constant or a shared variable in q, then $\pi \neq \pi_\sigma$.*

Let us now introduce the notion of *factorizability* which, as we explain below, makes one of the main differences between our algorithm and the one presented in [14], due to which a perfect rewriting with less BCQs is obtained.

Definition 2 (Factorizability) *Consider a BCQ q over a schema $\mathcal{R}$, and a TGD σ over $\mathcal{R}$ which contains an existentially quantified variable. A set of atoms $A \subseteq body(q)$, where $|A| \geqslant 2$, that unifies is* factorizable *w.r.t. σ if there exists a variable V that occurs in every atom of S only at position π_σ, and also V does not occur in $body(q) \setminus S$.*

It is important to clarify that in the case of (non-Boolean) CQs, the notion of factorizability is defined as above, except that the variable V does not occur in $(\{head(\sigma)\} \cup body(\sigma)) \setminus S$.

Example 1 (Factorization) Consider the BCQs

$$q_1 \ : \ q() \leftarrow \underbrace{t(A,B,C), t(A,E,C)}_{S_1}$$
$$q_2 \ : \ q() \leftarrow s(C), \underbrace{t(A,B,C), t(A,E,C)}_{S_2}$$
$$q_3 \ : \ q() \leftarrow \underbrace{t(A,B,C), t(A,C,C)}_{S_3}$$

and the TGD $\sigma : s(X), r(X,Y) \rightarrow \exists Z\, t(X,Y,Z)$. Clearly, S_1 is factorizable w.r.t. σ since the substitution $\{E \rightarrow B\}$ is a unifier for S_1, and also C appears in both atoms of S_1 only at position π_σ. The factorization results in the query $q() \leftarrow t(A,B,C)$; notice that σ is not applicable to S_1, but it is applicable to $\{t(A,B,C)\}$. On the contrary, despite the fact that S_2 unifies, it is not factorizable w.r.t. σ since C occurs also in $body(q_2) \setminus S_2$. Finally, even if S_3 unifies, it is not factorizable w.r.t. σ since C appears in S_3, not only at position π_σ, but also at position $t[2]$.

We are now ready to describe the algorithm TGD-rewrite, depicted in Algorithm 1, which is based on the rewriting algorithm presented in [14]. The perfect rewriting of a BCQ q w.r.t. a set of TGDs Σ is computed by exhaustively applying (i.e., until a fixpoint is reached) two steps: *factorization* and *rewriting*.

Algorithm 1: The algorithm TGD-rewrite

Input: a BCQ q over a schema $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$
Output: the FO-rewriting Q_{FIN} of q w.r.t. Σ

$Q_{\text{REW}} := \{\langle q, 1\rangle\}$;
repeat
  $Q_{\text{TEMP}} := Q_{\text{REW}}$;
  foreach $\{\langle q, x\rangle\} \in Q_{\text{TEMP}}$, where $x \in \{0, 1\}$, **do**
    /* factorization step */
    foreach $\sigma \in \Sigma$ **do**
      $q' := \mathit{factorize}(q, \sigma)$;
      if $\mathit{notExists}(\langle q', y\rangle, Q_{\text{REW}})$, where $y \in \{0, 1\}$, **then**
        $Q_{\text{REW}} := Q_{\text{REW}} \cup \{\langle q', 0\rangle\}$;
    /* rewriting step */
    foreach $A \subseteq \mathit{body}(q)$ **do**
      foreach $\sigma \in \Sigma$ **do**
        if $\mathit{isApplicable}(\sigma, A, q)$ **then**
          $q' := \gamma_{A \cup \{\mathit{head}(\sigma)\}}(q[A/\mathit{body}(\sigma)])$;
          if $\mathit{notExists}(\langle q', 1\rangle, Q_{\text{REW}})$ **then**
            $Q_{\text{REW}} := Q_{\text{REW}} \cup \{\langle q', 1\rangle\}$;
until $Q_{\text{TEMP}} = Q_{\text{REW}}$;
$Q_{\text{FIN}} := \{q \mid \langle q, x\rangle \in Q_{\text{REW}}$ and $x = 1\}$;
return Q_{FIN}

Factorization Step. The function $\mathit{factorize}(q, \sigma)$, providing that there exists a subset of $\mathit{body}(q)$ which is factorizable w.r.t. σ (otherwise, the query q is returned), first selects such a set $S \subseteq \mathit{body}(q)$. Then, the query q' is constructed by applying the MGU γ_S for S on q. Providing that there is no pair $\langle q'', y\rangle$, where $y \in \{0, 1\}$, in Q_{REW} such that q' and q'' are the same (modulo bijective variable renaming), the pair $\langle q', 0\rangle$ is added to Q_{REW}; the label 0 keeps track of the queries generated by the factorization step that must be excluded from the final rewriting. This is carried out by the *notExists* function.

Rewriting Step. If there exists a pair $\langle q, y\rangle$ and a TGD $\sigma \in \Sigma$ which is applicable to a set of atoms $A \subseteq \mathit{body}(q)$, then the algorithm constructs a new query $q' = \gamma_{A \cup \{\mathit{head}(\sigma)\}}(q[A/\mathit{body}(\sigma)])$, that is, the BCQ obtained from q by replacing A with $\mathit{body}(\sigma)$ and then applying the MGU for the set $A \cup \{\mathit{head}(\sigma)\}$. Providing that there is no pair $\langle q'', 1\rangle$ in Q_{REW} such that q' and q'' are the same (modulo bijective variable renaming), the pair $\langle q', 1\rangle$ is added to Q_{REW}; the label 1 keeps track of the queries generated by the rewriting step which will be the final rewriting.

Example 2 (Rewriting) Consider the set Σ of TGDs

$$\begin{array}{rcl} \sigma_1 & : & s(X) \rightarrow \exists Z \;\; t(X, X, Z) \\ \sigma_2 & : & t(X, Y, Z) \rightarrow r(Y, Z) \end{array}$$

and the query $q() \leftarrow t(A, B, C), r(B, C)$. TGD-rewrite first applies σ_2 to $\{r(B, C)\}$ since σ_1 is not applicable. The query $q_1 : q() \leftarrow t(A, B, C), t(V^1, B, C)$ is produced. Clearly, $\mathit{body}(q_1)$ is factorizable w.r.t. σ_1 and the query $q_2 : q() \leftarrow t(A, B, C)$ is obtained. Now, σ_1 is applicable to $\{t(A, B, C)\}$ and the query $q_3 : q() \leftarrow s(A)$ is obtained. The perfect rewriting constructed by the algorithm is the set $\{q, q_1, q_3\}$.

The next example shows that dropping the applicability condition, then TGD-rewrite may produce unsound rewritings.

Example 3 (Loss of soundness) Suppose that we ignore the applicability condition during the rewriting process. Consider the set Σ of TGDs given in Example 2, and also the BCQ $q_1 : q() \leftarrow$

$t(A, B, c)$, where c is a constant of Δ_c. A BCQ q' of the form $q() \leftarrow s(V)$ is obtained, where the information about the constant c is lost. Consider now the database $D = \{s(b), t(a, b, d)\}$ for $\mathcal{R}$. The query q' maps to the atom $s(b)$ which implies that $D \models q'$. However, the original query q does not map to $chase(D, \Sigma)$, and thus $D \cup \Sigma \not\models q$. Therefore, any rewriting containing q' is not a sound rewriting of q given Σ. Consider now the query $q'' : q() \leftarrow t(A, B, B)$. The same query q' mapping to the atom $s(b)$ of D is obtained. However, during the construction of $chase(D, \Sigma)$ it is not possible to get an atom of the form $t(X, Y, Y)$, where at positions $t[2]$ and $t[3]$ the same value occurs. This implies that there is no homomorphism that maps q to $chase(D, \Sigma)$, and hence $D \cup \Sigma \not\models q$. Therefore, any rewriting containing q' is again unsound.

The applicability condition may prevent the generation of queries that are vital to guarantee completeness of the rewritten query, as shown by the following example. This is exactly the reason why the factorization step is also needed.

Example 4 (Loss of completeness) Consider the set Σ of TGDs

$$\begin{array}{rcl} \sigma_1 & : & p(X) \rightarrow \exists Y\, t(X, Y) \\ \sigma_2 & : & t(X, Y) \rightarrow s(Y) \end{array}$$

and the query $q : q() \leftarrow t(A, B), s(B)$. The only viable strategy in this case is to apply σ_2 to $\{s(B)\}$, since σ_1 is not applicable to $\{t(A, B)\}$ due to the shared variable B. The query that we obtain is $q' : q() \leftarrow t(A, B), t(V^1, B)$, where V^1 is a fresh variable. Notice that in q' the variable B remains shared thus it is not possible to apply σ_1. It is obvious that without the factorization step there is no way to obtain the query $q'' : q() \leftarrow p(A)$ during the rewriting process. Now, consider the database $D = \{p(a)\}$. Clearly, $chase(D, \Sigma) = \{p(a), t(a, z_1), s(z_1)\}$, and therefore $chase(D, \Sigma) \models q$, or, equivalently, $D \cup \Sigma \models q$. However, the rewritten query is not entailed by the given database D, since q'' does not belong to it, which implies that it is not complete.

We proceed now to establish soundness and completeness of the proposed algorithm. Towards this aim we need two auxiliary technical lemmas. The first one, which is needed for soundness, states that once the chase entails the rewritten query constructed by the rewriting algorithm, then the chase entails also the given query. In the sequel, for brevity, given a BCQ q over a schema $\mathcal{R}$ and a set Σ of TGDs over $\mathcal{R}$, we denote by q_Σ the rewritten query $\mathsf{TGD\text{-}rewrite}(q, \Sigma)$.

Lemma 3 *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. If $chase(D, \Sigma) \models q_\Sigma$, then $chase(D, \Sigma) \models q$.*

Proof. The proof is by induction on the number of applications of the rewriting step. We denote by $q_\Sigma^{[i]}$ the part of the rewritten query q_Σ obtained by applying i times the rewriting step.

Base Step. Clearly, $q_\Sigma^0 = q_\Sigma$, and the claim holds trivially.

Inductive Step. Suppose now that $chase(D, \Sigma) \models q_\Sigma^{[i]}$, for $i \geq 0$. This implies that there exists $p \in q_\Sigma^{[i]}$ such that $chase(D, \Sigma) \models p$, and thus there exists a homomorphism h such that $h(body(p)) \subseteq chase(D, \Sigma)$. If $p \in q_\Sigma^{[i-1]}$, then the claim follows by induction hypothesis. The interesting case is when p was obtained during the i-th application of the rewriting step from a BCQ $p' \in q_\Sigma^{[i-1]}$, i.e., $q_\Sigma^{[i]} = q_\Sigma^{[i-1]} \cup \{p\}$. By induction hypothesis, it suffices to show that $chase(D, \Sigma) \models q_\Sigma^{[i-1]}$.

Clearly, there exists a TGD $\sigma \in \Sigma$ of the form $\phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z\, r(\mathbf{X}, Z)$ which is applicable to a set $A \subseteq body(p')$, and p is such that $body(p) = \gamma(p'[A/\phi(\mathbf{X}, \mathbf{Y})])$, where γ is the MGU for the set $A \cup \{head(\sigma)\}$. Observe that $h(\gamma(\phi(\mathbf{X}, \mathbf{Y}))) \subseteq chase(D, \Sigma)$, and hence σ is applicable to $chase(D, \Sigma)$; let $\mu = h \circ \gamma$. Thus, $\mu'(r(\mathbf{X}, Z)) \in chase(D, \Sigma)$, where $\mu' \supset \mu$. We define the substitution $h' = h \cup \{\gamma(Z) \rightarrow \mu'(Z)\}$.

Let us first show that h' is a well-defined substitution. It suffices to show that $\gamma(Z)$ is not a constant, and also that $\gamma(Z)$ does not appear in the left-hand side of an assertion of h. Towards a contradiction, suppose that $\gamma(Z)$ is either a constant or appears in the left-hand side of an assertion of h. It is easy to verify that in this case there exists an atom $\underline{a} \in A$ such that at position π_σ in $\underline{a}$ occurs either a constant or a variable which is shared in p'. But this contradicts the fact that σ is applicable to A. Consequently, h' is well-defined. It remains to show that the substitution $h' \circ \gamma$ maps $\mathit{body}(p')$ to $\mathit{chase}(D, \Sigma)$, and thus $\mathit{chase}(D, \Sigma) \models q_\Sigma^{[i-1]}$. Clearly, $\gamma(\mathit{body}(p') \setminus A) \subseteq \mathit{body}(p)$. Since $h(\mathit{body}(p)) \subseteq \mathit{chase}(D, \Sigma)$, we get that $h'(\gamma(\mathit{body}(p') \setminus A)) \subseteq \mathit{chase}(D, \Sigma)$. Moreover,

$$\begin{array}{rcl} h'(\gamma(A)) & = & h'(\gamma(r(\mathbf{X}, Z))) \\ & = & r(h'(\gamma(\mathbf{X})), h'(\gamma(Z))) \\ & = & r(\mu(\mathbf{X}), \mu'(Z)) \\ & = & \mu'(r(\mathbf{X}, Z)) \\ & \in & \mathit{chase}(D, \Sigma). \end{array}$$

The proof is now complete. □

The second auxiliary lemma, which is needed for completeness, asserts that once the chase entails the rewritten query constructed by the rewriting algorithm, then the given database also entails the rewritten query.

Lemma 4 *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. If $\mathit{chase}(D, \Sigma) \models q_\Sigma$, then $D \models q_\Sigma$.*

Proof. We proceed by induction on the number of applications of the chase step.

Base Step. Clearly, $\mathit{chase}^{[0]}(D, \Sigma) = D$, and the claim holds trivially.

Inductive Step. Suppose now that $\mathit{chase}^{[i]}(D, \Sigma) \models q_\Sigma$, for $i \geq 0$. This implies that there exists $p \in q_\Sigma$ such that $\mathit{chase}^{[i]}(D, \Sigma) \models p$, and thus there exists a homomorphism h such that $h(\mathit{body}(p)) \subseteq \mathit{chase}^{[i]}(D, \Sigma)$. If $h(\mathit{body}(p)) \subseteq \mathit{chase}^{[i-1]}(D, \Sigma)$, then the claim follows by induction hypothesis. The non-trivial case is when the atom $\underline{a}$, obtained during the i-th application of the chase step due to a TGD $\sigma \in \Sigma$ of the form $\phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z\, r(\mathbf{X}, Z)$, belongs to $h(\mathit{body}(p))$. Clearly, there exists a homomorphism μ such that $\mu(\phi(\mathbf{X}, \mathbf{Y})) \subseteq \mathit{chase}^{[i-1]}(D, \Sigma)$ and $\underline{a} = \mu'(r(\mathbf{X}, \mathbf{Y}))$, where $\mu' \supseteq \mu$. By induction hypothesis, it suffices to show that $\mathit{chase}^{[i-1]}(D, \Sigma) \models q_\Sigma$. Before we proceed further, we need to establish an auxiliary technical claim.

Claim 5 *There exists a BCQ $p' \in q_\Sigma$ and a set of atoms $A \subseteq \mathit{body}(p')$ such that σ is applicable to A, and also there exists a homomorphism λ such that $\lambda(\mathit{body}(p') \setminus A) \subseteq \mathit{chase}^{[i-1]}(D, \Sigma)$ and $\lambda(A) = \underline{a}$.*

Proof. Clearly, there exists a set of atoms B such that $h(\mathit{body}(p) \setminus B) \subseteq \mathit{chase}^{[i-1]}(D, \Sigma)$ and $h(B) = \underline{a}$. Observe that the null value that occurs in $\underline{a}$ at position π_σ does not occur in $\mathit{chase}^{[i-1]}(D, \Sigma)$ or in $\underline{a}$ at some position other than π_σ. Therefore, the variables that occur in the atoms of B at π_σ do not appear at some other position. Consequently, B can be partitioned into the sets $B_1, \ldots, B_m$, where $m \geq 1$, and the following holds: for each $i \in [m]$, in the atoms of B_i at position π_σ the same variable V_i occurs, and also V_i does not occur in some other set $B \in \{B_1, \ldots, B_m\} \setminus \{B_i\}$ or in B_i at some position other than π_σ. It is easy to verify that each set B_i is factorizable w.r.t. σ.

Suppose that we factorize B_1. Then, the query $p_1 = \gamma_1(p)$, where γ_1 is the MGU for B_1, is obtained. Observe that h is a unifier for B_1. By definition of the MGU, there exists a substitution θ_1 such that $h = \theta_1 \circ \gamma_1$. Clearly,

$$\begin{array}{rcl} \theta_1(\mathit{body}(p_1) \setminus \gamma_1(B)) & = & \theta_1(\gamma_1(\mathit{body}(p)) \setminus \gamma_1(B)) \\ & = & h(\mathit{body}(p) \setminus B) \\ & \subseteq & \mathit{chase}^{[i-1]}(D, \Sigma), \end{array}$$

and $\theta_1(\gamma_1(B)) = h(B) = \underline{a}$.

Now, observe that the set $\gamma_1(B_2) \subseteq \mathit{body}(p_1)$ is factorizable w.r.t. σ. By applying factorization we get the query $p_2 = \gamma_2(p_1)$, where γ_2 is the MGU for $\gamma_1(B_2)$. Since θ_1 is a unifier for $\gamma_1(B_2)$, there exists a substitution θ_2 such that $\theta_1 = \theta_2 \circ \gamma_2$. Clearly,

$$\begin{array}{rcl} \theta_2(\mathit{body}(p_2) \setminus \gamma_2(\gamma_1(B))) & = & \theta_2(\gamma_2(\mathit{body}(p_1)) \setminus \gamma_2(\gamma_1(B))) \\ & = & \theta_1(\gamma_1(\mathit{body}(p)) \setminus \gamma_1(B)) \\ & = & h(\mathit{body}(p) \setminus B) \\ & \subseteq & \mathit{chase}^{[i-1]}(D, \Sigma), \end{array}$$

and $\theta_2(\gamma_2(\gamma_1(B))) = \theta_1(\gamma_1(B)) = h(B) = \underline{a}$.

Eventually, by applying the factorization step as above, we will get the BCQ

$$p_m \ = \ \gamma_m \circ \ldots \circ \gamma_1(p),$$

where γ_j is the MGU for the set $\gamma_{j-1} \circ \ldots \circ \gamma_1(B_j)$, for $j \in \{2, \ldots, m\}$ (recall that γ_1 is the MGU for B_1), such that $\theta_m(\mathit{body}(p_m) \setminus \gamma_m \circ \ldots \circ \gamma_1(B)) \subseteq \mathit{chase}^{[i-1]}(D, \Sigma)$ and $\theta_m(\gamma_m \circ \ldots \circ \gamma_1(B)) = \underline{a}$.

It is easy to verify that σ is applicable to A. The claim follows with $p' = p_m$, $A = \gamma_m \circ \ldots \circ \gamma_1(B)$ and $\lambda = \theta_m$. □

The above claim implies that during the rewriting process eventually we will get a BCQ p'' such that $\mathit{body}(p'') = \gamma(\mathit{body}(p') \setminus A) \cup \gamma(\phi(\mathbf{X}, \mathbf{Y}))$, where γ is the MGU for the set $A \cup \{\mathit{head}(\sigma)\}$. It remains to show that there exists a homomorphism that maps $\mathit{body}(p'')$ to $\mathit{chase}^{[i-1]}(D, \Sigma)$. Since $\lambda \cup \mu'$ is a well-defined substitution, we get that $\lambda \cup \mu'$ is a unifier for $A \cup \{\mathit{head}(\sigma)\}$. By definition of the MGU, there exists a substitution θ such that $\lambda \cup \mu' = \theta \circ \gamma$. Observe that

$$\begin{array}{rcl} \theta(\mathit{body}(p'')) & = & \theta(\gamma(\mathit{body}(p') \setminus A) \cup \gamma(\phi(\mathbf{X}, \mathbf{Y}))) \\ & = & (\lambda \cup \mu')(\mathit{body}(p') \setminus A) \cup (\lambda \cup \mu')(\phi(\mathbf{X}, \mathbf{Y})) \\ & = & \lambda(\mathit{body}(p') \setminus A) \cup \mu'(\phi(\mathbf{X}, \mathbf{Y})) \\ & \subseteq & \mathit{chase}^{[i-1]}(D, \Sigma). \end{array}$$

Consequently, the desired homomorphism is θ, and the claim follows. □

We are now ready to establish soundness and completeness of the algorithm TGD-rewrite.

Theorem 6 *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. It holds that, $D \models q_\Sigma$ iff $D \cup \Sigma \models q$.*

Proof. Suppose first that $D \models q_\Sigma$. Since $D \subseteq \mathit{chase}(D, \Sigma)$, we get that $\mathit{chase}(D, \Sigma) \models q_\Sigma$, and the claim follows by Lemma 3. Suppose now that $D \cup \Sigma \models q_\Sigma$. Since $q \in q_\Sigma$, we get that $\mathit{chase}(D, \Sigma) \models q_\Sigma$, and the claim follows by Lemma 4. □

Notice that the above result holds for arbitrary TGDs. However, termination of TGD-rewrite is guaranteed if we consider linear, sticky or sticky-join sets of TGDs since, during the rewriting process, only finitely many queries (modulo bijective variable renaming) are generated.

Theorem 7 *The algorithm TGD-rewrite terminates under linear, sticky or sticky-join sets of TGDs.*

Approaches such as those of [5] and [14] resort to exhaustive factorizations of the atoms in the queries generated by the rewriting algorithm. By factorizing a query q we obtain a subquery q', that is, q implies q' (w.r.t. the given set of TGDs). Observe that by computing the factorized query q' we eliminate unnecessary shared variables, in the body of q, due to which the applicability condition is violated. Consider for example the query q' of Example 4. By factorizing the body of q' we obtain the query $q() \leftarrow t(A, B)$ which is a subquery (w.r.t. to the given set Σ of TGDs)

of q' (in this case equivalent to q'), where the variable B is no longer shared. Thus, the rewriting step can now apply σ_1 to $\{t(A, B)\}$ and produce the query $q() \leftarrow p(A)$ which is needed to ensure completeness.

The exhaustive factorization produces a non-negligible number of redundant queries as demonstrated by the simple example above. It is thus necessary to apply a restricted form of factorization that generates a possibly small number of BCQs that are necessary to guarantee completeness of the rewritten query. This corresponds to the identification of all the atoms in the query whose shared existential variables come from the same atom in the chase, and they can be thus unified with no loss of information. The key principle behind our factorization process is that, in order to be applied, there must exist a TGD that can be applied to the output of the factorization.

5.1 Exploiting Negative Constraints

It is well-known that negative constraints (NCs) of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$ are vital for representing ontologies. As already explained in Subsection 4.2, given a database D for a schema $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$, and a set $\Sigma_\bot$ of NCs over $\mathcal{R}$, once the theory $D \cup \Sigma \cup \Sigma_\bot$ is consistent, then we are allowed to ignore the NCs since, for every BCQ q, $D \cup \Sigma \cup \Sigma_\bot \models q$ iff $D \cup \Sigma \models q$. However, as shown in the following example, by exploiting the given set of NCs it is possible to further reduce the size of the final rewriting.

Example 5 *Consider the TGD $\sigma : t(X), s(Y) \rightarrow \exists Z\, p(Y, Z)$, the NC $\nu : r(X, Y), s(Y) \rightarrow \bot$, and the BCQ $q() \leftarrow r(A, B), p(B, C)$. Clearly, due to the rewriting step, the query $p : q() \leftarrow r(A, B), t(V^1), s(B)$ is obtained during the rewriting process. However, this query is not really needed since, for any database D for $\mathcal{R}$, $D \not\models p$; otherwise, D violates the NC ν which is a contradiction since we always assume that the theory $D \cup \{\sigma, \nu\}$ is consistent.*

It is not difficult to show that, given a BCQ q, and a set Σ of TGDs, if a query $p \in q_\Sigma$ is not entailed by $chase(D, \Sigma)$, for an arbitrary database D, then any query $p' \in q_\Sigma$ obtained during the rewriting process starting from p, also it is not entailed by $chase(D, \Sigma)$. Assume now that the set $\Sigma_\bot$ of NCs is part of the input. If we obtain a query $p \in q_\Sigma$ such that there exists a homomorphism that maps $body(\nu)$, for some NC $\nu \in \Sigma_\bot$, to $body(p)$, then we can safely ignore p since $chase(D, \Sigma)$ does not entail p.

From the above informal discussion, we conclude that we can further reduce the size of the final rewriting by modifying our algorithm as follows. During the execution of the rewriting algorithm TGD-rewrite (see Algorithm 1), after the factorization step (resp., rewriting step) we check whether there exists a homomorphism that maps $body(\nu)$, for some NC ν of the given set of NCs, to the body of the generated query q'. If there exists such a homomorphism, then the pair $\langle q', 0\rangle$ (resp., $\langle q', 1\rangle$) is not added to the set Q_{REW}. Furthermore, the pair $\langle q, 1\rangle$ is added to Q_{REW} (see the first line of the algorithm) only if there is no homomorphism that maps $body(\nu)$, for some NC ν of the given set of NCs, to $body(q)$. If there exists such a homomorphism, then the algorithm terminates and returns the emptyset, which means that $chase(D, \Sigma) \not\models q$, for every database D for $\mathcal{R}$.

6 Rewriting Optimization

It is common knowledge that the perfect rewriting obtained by applying a backward-chaining rewriting algorithm (like TGD-rewrite) is, in general, not very well-suited for execution by a DB engine due to the large number of queries to be evaluated. In this section we propose a technique, called *query elimination*, aiming at optimizing the obtained rewritten query under the class of linear TGDs. As we shall see, query elimination (which is an additional step during the execution of the algorithm TGD-rewrite) reduces *(i)* the number of BCQs of the perfect rewriting, *(ii)* the number of atoms in each query of the rewriting as well as *(iii)* the number of joins. Note that in the rest of

the paper we restrict our attention on linear TGDs. Recall that linear TGDs are TGDs with just one atom in their body. Since we also assume, as explained in the previous section, TGDs with just one atom in their head, henceforth, when using the term TGD, we shall refer to TGDs with just one body-atom and one head-atom.

By exploiting the given set of TGDs, it is possible to identify atoms in the body of a certain query that are logically implied (w.r.t. the given set of TGDs) by other atoms in the same query. In particular, for each BCQ q obtained by applying the rewriting step of TGD-rewrite, the atoms of $body(q)$ that are logically implied (w.r.t. the given set of TGDs) by some other atom of $body(q)$ are eliminated. Roughly speaking, the elimination of an atom from the body of a query implies the avoidance of the construction of redundant queries during the rewriting process. Thus, this step greatly reduces the number of BCQs in the perfect rewriting. Before going into the details, let us first introduce some necessary technical notions.

Definition 3 (Dependency Graph) *Consider a set Σ of TGDs over a schema $\mathcal{R}$. The* dependency graph *of Σ is a labeled directed multigraph $\langle N, E, \lambda \rangle$, where N is the node set, E is the edge set, and λ is a labeling function $E \to \Sigma$. The node set N is the set of positions of $\mathcal{R}$. If there is a TGD $\sigma \in \Sigma$ such that the same variable appears at position π_b in $body(\sigma)$ and at position π_h in $head(\sigma)$, then in E there is an edge $e = (\pi_b, \pi_h)$ with $\lambda(e) = \sigma$.*

Intuitively speaking, the dependency graph of a set Σ of TGDs describes all the possible ways of propagating a term from a position to some other position during the construction of the chase under Σ. More precisely, the existence of a path P from π_1 to π_2 implies that it is possible (but not always) to propagate a term from π_1 to π_2. The existence of P guarantees the propagation of a term from π_1 to π_2 if, for each pair of consecutive edges $e = (\pi, \pi')$ and $e' = (\pi', \pi'')$ of P, where e and e' are labeled by the TGDs σ and σ', respectively, the atom obtained during the chase by applying σ triggers σ'. To verify whether this holds we need an additional piece of information, the so-called *equality type*, about the body-atom and the head-atom of each TGD that occurs in P.

Definition 4 (Equality Type) *Consider an atom $\underline{a}$ of the form $r(t_1, \ldots, t_n)$, where $n \geq 1$. The* equality type *of $\underline{a}$ is the set of equalities*

$$\{r[i] = r[j] \mid t_i, t_j \notin \Delta_c \text{ and } t_i = t_j\} \bigcup \{r[i] = c \mid c \in \Delta_c \text{ and } t_i = c\}.$$

We denote the above set as $eq(\underline{a})$.

It is straightforward to see that, given a pair of TGDs σ and σ', if $eq(body(\sigma')) \subseteq eq(head(\sigma))$, then there exists a substitution μ such that $\mu(body(\sigma')) = head(\sigma)$. This allows us to show that the atom obtained by applying σ during the construction of the chase triggers σ'. Consequently, the existence of a path P (as above) guarantees the propagation of a term from π_1 to π_2 if, for each pair of consecutive edges e and e' of P which are labeled by σ and σ', respectively, $eq(body(\sigma')) \subseteq eq(head(\sigma))$.

Example 6 (*Dependency Graph)* Consider the set Σ of TGDs

$$\begin{array}{lcl} \sigma_1 & : & p(X,Y) \to \exists Z r(X,Y,Z) \\ \sigma_2 & : & r(X,Y,c) \to s(X,Y,Y) \\ \sigma_3 & : & s(X,X,Y) \to p(X,Y). \end{array}$$

The equality type of the body-atoms and head-atoms of the TGDs of Σ are as follows:

$$\begin{array}{lcl} eq(body(\sigma_1)) & = & \varnothing \\ eq(head(\sigma_1)) & = & \varnothing \\ eq(body(\sigma_2)) & = & \{r[3] = c\} \\ eq(head(\sigma_2)) & = & \{s[2] = s[3]\} \\ eq(body(\sigma_3)) & = & \{s[1] = s[2]\} \\ eq(head(\sigma_3)) & = & \varnothing. \end{array}$$

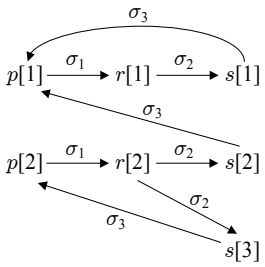


Figure 2: Dependency graph for Example 6.

The dependency graph of Σ is shown in Figure 2.

We are now ready, by exploiting the dependency graph of a set of TGDs, and the equality type of an atom, to introduce *atom coverage*.

Definition 5 (Atom Coverage) *Consider a BCQ q over a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. Let $\underline{a}$ and $\underline{b}$ be atoms of $body(q)$, where $\{t_1, \ldots, t_n\}$, for $n \geq 0$, is the set of shared variables and constants that occur in $\underline{b}$. Also, let G_Σ be the dependency graph of Σ. We say that $\underline{a}$* covers *$\underline{b}$ w.r.t. q and Σ, written as $\underline{a} \prec^q_\Sigma \underline{b}$, if for each $i \in [n]$: (i) the term t_i occurs also in $\underline{a}$, and (ii) if t_i occurs in $\underline{a}$ and $\underline{b}$ at positions $\Pi_{\underline{a},i}$ and $\Pi_{\underline{b},i}$, respectively, then, there exists an integer $k \geq 2$ and a set of TGDs $\{\sigma_1, \ldots, \sigma_{k-1}\} \subseteq \Sigma$, where $eq(body(\sigma_1)) \subseteq eq(\underline{a})$ and, for each $j \in [k-2]$, $eq(body(\sigma_{j+1})) \subseteq eq(head(\sigma_j))$, such that, for each $\pi \in \Pi_{\underline{b},i}$, in G_Σ there exists a path $\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}$, where $\pi_{i_1} \in \Pi_{\underline{a},i}$, $\pi_{i_k} = \pi$, and $\lambda((\pi_{i_j}, \pi_{i_{j+1}})) = \sigma_j$, for each $j \in [k-1]$.*

Condition *(i)* ensures that by removing $\underline{b}$ from q we do not loose any constant, and also all the joins between $\underline{b}$ and the other atoms of $body(q)$, except $\underline{a}$, are preserved. Condition *(ii)* guarantees that the atom $\underline{b}$ is logically implied (w.r.t. Σ) by the atom $\underline{a}$, and therefore can be eliminated.

Lemma 8 *Consider a BCQ q over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Suppose that $\underline{a} \prec^q_\Sigma \underline{b}$, where $\underline{a}, \underline{b} \in body(q)$, and q' is the BCQ obtained from q by eliminating the atom $\underline{b}$. Then, $I \models q$ iff $I \models q'$, for each instance I that satisfies Σ.*

Proof (Sketch). ($\Rightarrow$) By hypothesis, there exists a homomorphism h such that $h(body(q)) \subseteq I$. Since, by definition of q', $body(q') \subset body(q)$, we immediately get that $h(body(q')) \subseteq I$, which implies that $I \models q'$.

($\Leftarrow$) Conversely, there exists a homomorphism h such that $h(body(q')) \subseteq I$, and thus $h(body(q) \setminus \{\underline{b}\}) \subseteq I$. It suffices to show that there exists an extension of h which maps $\underline{b}$ to I. Since $\underline{a} \prec^q_\Sigma \underline{b}$, it is not difficult to verify that there exists an atom $\underline{c} \in I$ such that $eq(\underline{b}) = eq(\underline{c})$, which implies that there exists a substitution μ such that $\mu(\underline{b}) = \underline{c}$, and also μ is compatible with h. Consequently, $(h \cup \mu)(body(q)) \subseteq I$, and thus $I \models q$. □

An *atom elimination strategy* for a BCQ is a permutation of its body-atoms. Given a BCQ q and a set Σ of linear TGDs, the set of atoms of $body(q)$ that cover $\underline{a} \in body(q)$ w.r.t. Σ, denoted as $cover(\underline{a}, q, \Sigma)$, is the set $\{\underline{b} \mid \underline{b} \in body(q) \text{ and } \underline{b} \prec^q_\Sigma \underline{a}\}$; when q and Σ are obvious from the context, we shall denote the above set as $cover(\underline{a})$. By exploiting the cover set of the atoms of $body(q)$, we associate to each atom elimination strategy S for q a subset of $body(q)$, denoted $eliminate(q, S, \Sigma)$, which is the set of atoms of $body(q)$ that can be safely eliminated (according to S) in order to obtain a logically equivalent query (w.r.t. Σ) with less atoms in its body. Formally, $eliminate(q, S, \Sigma)$ is computed by applying the following procedure; in the sequel, let $S = [\underline{a}_1, \ldots, \underline{a}_n]$, where $\{\underline{a}_1, \ldots, \underline{a}_n\} = body(q)$:

$A := \varnothing$;

foreach $i := 1$ to n **do**
  $\underline{a} := S[i]$;
  if $cover(\underline{a}) \neq \varnothing$ **then**
    $A := A \cup \{\underline{a}\}$;
    foreach $\underline{b} \in body(q) \setminus A$ **do**
      $cover(\underline{b}) := cover(\underline{b}) \setminus \{\underline{a}\}$;
return A.

By exploiting the fact that the binary relation $\prec_{\Sigma}^{q}$ is transitive, it is possible to establish the uniqueness (w.r.t. the number of the eliminated atoms) of the atom elimination strategy for a BCQ. In particular, the following lemma can be shown.

Lemma 9 *Consider a BCQ q over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Let S_1 and S_2 be arbitrary elimination strategies for q. It holds that, $|eliminate(q, S_1, \Sigma)| = |eliminate(q, S_2, \Sigma)|$.*

Since the elimination strategy for a query is unique (w.r.t. the number of the eliminated atoms), in the rest of this section we refer to the set of atoms that can be safely eliminated from a query q (w.r.t. a set Σ of linear TGDs) by $eliminate(q, \Sigma)$.

We are now ready to describe how query elimination works. During the execution of the rewriting algorithm TGD-rewrite (see Algorithm 1), after the factorization step and the rewriting step the so-called *elimination* step is applied. In particular, the factorized query q' obtained during the factorization step is the query $eliminate(factorize(q, \sigma), \Sigma)$, while the rewritten query obtained during the rewriting step is the query $eliminate(\gamma_{A \cup \{head(\sigma)\}}(q[A/body(\sigma)]), \Sigma)$. Moreover, instead of adding the given query q in Q_{REW}, we add the eliminated query. In particular, the first line of the algorithm is replaced by $Q_{\text{REW}} := \langle eliminate(q), 1 \rangle$. An example of query elimination follows.

Example 7 (*Query Elimination)* Consider the set Σ of TGDs of Example 6, and the BCQ

$$q() \leftarrow \underbrace{p(A, B)}_{\underline{a}}, \underbrace{r(A, B, C)}_{\underline{b}}, \underbrace{s(A, A, D)}_{\underline{c}}.$$

Based on the Definition 5, it is an easy task to verify that $cover(\underline{a}) = \varnothing$, $cover(\underline{b}) = \{\underline{a}\}$ and $cover(\underline{c}) = \varnothing$. Therefore, the output of the function $eliminate(q, \Sigma)$ is the singleton set $\{\underline{b}\}$. Consequently, by applying the elimination step we get the BCQ $q() \leftarrow p(A, B), s(A, A, D)$.

As already mentioned, the fact that an atom $\underline{a}$ covers some atom $\underline{b}$, means that $\underline{b}$ is logically implied (w.r.t. the given set of TGDs) by $\underline{a}$. However, as shown by the following example, this fact is not also necessary for the implication of $\underline{b}$ by $\underline{a}$.

Example 8 (*Atom Implication)* Consider the set Σ of TGDs of Example 6, and the BCQ q

$$q() \leftarrow \underbrace{r(A, A, c)}_{\underline{a}}, \underbrace{p(A, A)}_{\underline{b}},$$

where c is a constant of Δ_c. Observe that $\underline{a}$ does not cover $\underline{b}$ since, despite the existence of the paths $r[1]s[1]p[1]$ and $r[2]s[3]p[2]$ in the dependency graph of Σ, $eq(body(\sigma_3)) \not\subseteq eq(head(\sigma_2))$. However, $\underline{b}$ is logically implied (w.r.t. Σ) by $\underline{a}$. In particular, for every instance I that satisfies Σ, if $I \models \underline{a}$, which implies that an atom of the from $r(V, V, c)$ exists in I, then due to the TGDs σ_2 and σ_3 there exists also an atom $p(V, V)$, and thus $I \models \underline{b}$. Note that such cases are identified by the C&B algorithm [15]. Nevertheless, as already criticized in Section 2, this requires to pay a price in the number of queries in the rewritten query.

It is not difficult to see that the function *eliminate* runs in quadratic time in the number of atoms of $body(q)$ (by considering the given set of TGDs as fixed). In particular, to compute the cover set of each body-atom of q we need to consider all the pairs of atoms of $body(q)$. Note that the problem whether a certain atom $\underline{a}$ covers some other atom $\underline{b}$ is feasible in constant time since the given set of TGDs (and thus its dependency graph) is fixed.

The following result implies that the rewriting algorithm TGD-rewrite*, obtained from TGD-rewrite by applying the additional step of elimination, is still sound and complete.

Theorem 10 *Consider a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$. Then, $D \models$ TGD-rewrite$^\star(\mathcal{R}, \Sigma, q)$ iff $D \cup \Sigma \models q$.*

Proof (Sketch). This result follows from the fact that the algorithm TGD-rewrite is sound and complete under linear TGDs (see Theorem 6) and Lemma 8. □

It is important to clarify that the above result does not hold if we consider arbitrary TGDs (as in Theorem 6). This is because the proof of Lemma 8, which states that atom coverage implies logical implication (w.r.t. the given set of TGDs), is based heavily on the linearity of TGDs. Termination of TGD-rewrite* follows immediately from the fact that TGD-rewrite terminates under linear TGDs (see Theorem 7).

7 Implementation and Experimental Setting

TGD-rewrite (without the additional check described in Subsection 5.1) and the query elimination technique presented in Section 6 have been implemented in the prototype system Nyaya [36] available at `http://mais.dia.uniroma3.it/Nyaya`. The reasoning and query answering engine is based on the IRIS Datalog engine[7] extended to support the FO-rewritable fragments of the Datalog$^\pm$ family. In particular, we extended IRIS to natively support existential variables in the head without introducing function symbols and to support the constant *false* as head of a rule (used to represent negative constraints). Both IRIS and our extension are implemented in Java.

Since TGD-rewrite is designed for reasoning over ontologies with large ABoxes, we put ourselves in a similar experimental setting such that of [19]. Thus, we use DL-Lite$_\mathcal{R}$ ontologies with a varying number of axioms. The queries under consideration are based on canonical examples used in the research projects where these ontologies have been developed. VICODI (V) is an ontology of European history, and developed in the EU-funded VICODI project[8]. STOCKEXCHANGE (S) is an ontology for representing the domain of financial institutions of the European Union. UNIVERSITY (U) is a DL-Lite$_\mathcal{R}$ version of the LUBM Benchmark[9], developed at Lehigh University, and describes the organizational structure of universities. ADOLENA (A) (Abilities and Disabilities OntoLogy for ENhancing Accessibility) is an ontology developed for the South African National Accessibility Portal, and describes abilities, disabilities and devices. The Path5 (P5) ontology is a synthetic ontology encoding graph structures and used to generate an exponential-blowup of the size of the rewritten queries. Recall that the transformation of a set of TGDs into an equivalent set of single-head TGDs with a single existential variable can introduce auxiliary predicates and rules (see Lemmas 1 and 2). The ontologies UX, AX and P5X are equivalent ontologies to U, A and P5 where the auxiliary predicates are considered part of the schema. These ontologies allow to study the impact of such transformations on the size of the rewriting.

We compared our implementation with two other rewriting-based query answering systems for FO-rewritable ontologies: QuOnto[10], based on [5] and developed by the University of Rome La Sapienza, and Requiem[11], based on [19] and developed by the Knowledge Representation group of the University of Oxford.

[7] `http://www.iris-reasoner.org/`.
[8] `http://www.vicodi.org`.
[9] `http://swat.cse.lehigh.edu/projects/lubm/`.
[10] `http://www.dis.uniroma1.it/quonto/`.
[11] `http://www.comlab.ox.ac.uk/projects/requiem/home.html`.

Table 1: Evaluation of Nyaya System.

		Size				Length				Width			
		QO	RQ	NY	NY⋆	QO	RQ	NY	NY⋆	QO	RQ	NY	NY⋆
V	q_1	15	15	15	15	15	15	15	15	0	0	0	0
	q_2	11	10	10	10	32	30	30	30	31	30	30	30
	q_3	72	72	72	72	216	216	216	216	144	144	144	144
	q_4	185	185	185	185	555	555	555	555	370	370	370	370
	q_5	150	30	30	30	900	210	210	210	1,110	270	270	270
S	q_1	6	6	6	6	6	6	6	6	0	0	0	0
	q_2	204	160	160	2	566	480	480	2	362	320	320	0
	q_3	1,194	480	480	4	5,026	2,400	2,400	8	4,778	2,400	2,400	4
	q_4	1,632	960	960	4	7,384	4,800	4,800	8	7,112	4,800	4,800	4
	q_5	11,487	2,880	2,880	8	67,664	20,160	20,160	24	84,064	25,920	25,920	24
U	q_1	5	2	2	2	10	4	4	4	5	2	2	2
	q_2	287	148	148	1	813	444	444	1	526	296	296	0
	q_3	1,260	224	224	4	7,296	1,344	1,344	16	10,812	2,016	2,016	20
	q_4	5,364	1,628	1,628	2	15,723	4,884	4,884	2	10,393	3,256	3,256	0
	q_5	9,245	2,960	2,960	10	35,710	11,840	11,840	20	52,970	17,760	17,760	20
A	q_1	783	402	402	247	1,540	779	779	197	757	377	377	86
	q_2	1,812	103	103	92	5,350	256	256	234	3,538	153	153	142
	q_3	4,763	104	104	104	23,804	520	520	520	23,804	520	520	520
	q_4	7,251	492	492	454	21,406	1,288	1,288	1,212	14,155	796	796	758
	q_5	66,068	624	624	624	195,042	3,120	3,120	3,120	128,974	3,120	3,120	3,120
P5	q_1	14	6	6	6	14	6	6	6	0	0	0	0
	q_2	86	10	10	10	156	16	16	16	70	6	6	6
	q_3	538	13	13	13	1,413	29	29	29	900	16	16	16
	q_4	3,620	15	15	15	14,430	44	44	44	10,260	29	29	29
	q_5	25,256	16	16	16	107,484	60	60	60	103,361	44	44	44
UX	q_1	5	5	5	5	10	10	10	10	5	5	5	5
	q_2	286	240	240	1	156	147	147	1	70	70	70	0
	q_3	1,248	1,008	1,008	12	1,397	1,125	1,125	48	892	735	735	60
	q_4	5,358	5,000	5,000	5	12,006	7,578	7,578	5	9,828	5,625	5,625	0
	q_5	9,220	8,000	8,000	25	101,652	47,656	47,656	50	96,677	37,890	37,890	50
AX	q_1	783	782	782	555	1,543	1,541	1,541	1,084	763	761	761	529
	q_2	1,812	1,781	1,781	1,737	3,589	3,528	3,528	3,514	3,576	3,516	3,516	3,401
	q_3	4,763	4,752	4,752	4,741	27,705	23,760	23,760	23,760	23,824	23,815	23,815	23,694
	q_4	7,251	7,100	7,100	6,564	7,739	7,578	7,578	6,178	5,744	5,625	5,625	5,201
	q_5	-	-	76,032	76,032	-	-	81,173	81,173	-	-	95,942	95,942
P5X	q_1	14	14	14	14	14	14	14	14	0	0	0	0
	q_2	86	77	77	66	156	147	147	121	70	70	70	55
	q_3	530	390	390	329	1,397	1,125	1,125	925	892	735	735	596
	q_4	3,476	1,953	1,953	1,644	12,006	7,578	7,578	6,263	9,828	5,625	5,625	4,619
	q_5	23,744	9,766	9,766	8,219	101,652	47,656	47,656	39,531	96,677	37,890	37,890	31,312

Since TGD-rewrite, as well as the algorithms presented in [5] and [19], are proven to be sound and complete, the most relevant way of judging the quality of the rewriting is the *size* of the perfect rewriting, i.e., the number of CQs in the perfect UCQ rewriting. In addition, we use two additional metrics, namely, the *length* of the rewriting, i.e., the number of atoms in the perfect rewriting, and the *width*, i.e., the number of joins to be performed when the rewritten query is executed. We believe these metrics to be more appropriate than the number of symbols in the rewritten query used, for example, in [19], since they allow to establish in a more precise way the cost of executing the rewriting on a database system. Table 1 reports the results of our experiments[12] while Table 2 shows the queries used in the experiments. We use the symbol "-" to denote those cases where the tool did not complete the rewriting within 15 minutes. By QO and RQ we refer to the QuOnto and Requiem systems, respectively, while NY and NY⋆ refer to Nyaya with factorisation and Nyaya with both factorisation and query elimination, respectively. All the tests have been performed on an Intel Core 2 Duo Processor at 2.50 GHz and 4GB of RAM. The OS is Ubuntu Linux 9.10 carrying a Sun JVM Standard Edition with maximum heap size set at 2GB of RAM.

As it can be seen, query elimination provides a substantial advantage in terms of the size of the perfect rewriting for the real-world ontologies A, U and S. In particular, for the queries denoted as Q2 in U and S, our procedure eliminates all the redundant atoms in the input query, and drastically reduces the number of queries in the final rewriting. On the other side, query elimination is not particularly effective in the synthetic test case P5 and P5X, since these cases have been intentionally created in order to generate perfect rewritings of exponential size.

8 Future Work

We plan to investigate rewriting and optimization techniques for sticky-join sets of TGDs, and alternative forms of rewriting such as positive-existential queries. We also plan to develop improved

[12] Additional data can be found on the Nyaya's Web site.

Table 2: Test Queries

TBox	Queries
V	$q_1(A) \leftarrow Location(A)$. $q_2(A, B) \leftarrow Military\_Person(A), hasRole(B, A), related(A, C)$. $q_3(A, B) \leftarrow Time\_Dependant\_Relation(A), hasRelationMember(A, B), Event(B)$. $q_4(A, B) \leftarrow Object(A), hasRole(A, B), Symbol(B)$. $q_5(A) \leftarrow Individual(A), hasRole(A, B), Scientist(B), hasRole(A, C), Discoverer(C), hasRole(A, D), Inventor(D)$.
S	$q_1(A) \leftarrow StockExchangeMember(A)$. $q_2(A, B) \leftarrow Person(A), hasStock(A, B), Stock(B)$. $q_3(A, B, C) \leftarrow FinantialInstrument(A), belongsToCompany(A, B), Company(B), hasStock(B, C), Stock(C)$. $q_4(A, B, C) \leftarrow Person(A), hasStock(A, B), Stock(B), isListedIn(B, C), StockExchangeList(C)$. $q_5(A, B, C, D) \leftarrow FinantialInstrument(A), belongsToCompany(A, B), Company(B), hasStock(B, C), Stock(C)$, $isListedIn(B, D), StockExchangeList(D)$.
U(X)	$q_1(A) \leftarrow worksFor(A, B), affiliatedOrganizationOf(B, C)$. $q_2(A, B) \leftarrow Person(A), teacherOf(A, B), Course(B)$. $q_3(A, B, C) \leftarrow Student(A), advisor(A, B), FacultyStaff(B), takesCourse(A, C), teacherOf(B, C), Course(C)$. $q_4(A, B) \leftarrow Person(A), worksFor(A, B), Organization(B)$. $q_5(A) \leftarrow Person(A), worksFor(A, B), University(B), hasAlumnus(B, A)$.
A(X)	$q_1(A) \leftarrow Device(A), assistsWith(A, B)$. $q_2(A) \leftarrow Device(A), assistsWith(A, B), UpperLimbMobility(B)$. $q_3(A) \leftarrow Device(A), assistsWith(A, B), Hear(B), affects(C, B), Autism(C)$. $q_4(A) \leftarrow Device(A), assistsWith(A, B), PhysicalAbility(B)$. $q_5(A) \leftarrow Device(A), assistsWith(A, B), PhysicalAbility(B), affects(C, B), Quadriplegia(C)$.
P5(X)	$q_1(A) \leftarrow edge(A, B)$. $q_2(A) \leftarrow edge(A, B), edge(B, C)$. $q_3(A) \leftarrow edge(A, B), edge(B, C), edge(C, D)$. $q_4(A) \leftarrow edge(A, B), edge(B, C), edge(C, D), edge(D, E)$. $q_4(A) \leftarrow edge(A, B), edge(B, C), edge(C, D), edge(D, E), edge(E, F)$.

techniques for rewriting an ontological query into a non-recursive Datalog program, rather than into a union of conjunctive queries (recall the discussion in Section 2). While the current approaches yield exponentially large non-recursive Datalog programs, it is possible to rewrite queries and TBoxes into non-recursive Datalog programs whose size is simultaneously polynomial in the query and the TBox. This will be dealt in a forthcoming paper.

Acknowledgments

G. Gottlob's work was funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC grant no. 246858 – DIADEM. Gottlob gratefully acknowledges a Royal Society Wolfson Research Merit Award. G. Orsi and G. Gottlob also acknowledge the Oxford Martin School - Institute for the Future of Computing. A. Pieris' work was funded by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). We thank Michaël Thomazo for his useful and constructive comments on the conference version of this paper.

References

[1] G. Gottlob, G. Orsi, and A. Pieris, "Ontological queries: Rewriting and optimization," in *Proc. of the 27th Intl Conf. on Data Engineering (ICDE)*, 2011, pp. 2–13.

[2] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[3] Wikipedia, "Ontology (information science)," 2010. [Online]. Available: {http://en.wikipedia.org/wiki/Ontology_(information_science)}.

[4] D. McComb, "The enterprise ontology," 2006. [Online]. Available: {http://www.tdan.com/view-articles/5016}.

[5] D. Calvanese, G. de Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The DL-Lite Family," *Journal of Automated Reasoning*, vol. 39, no. 3, pp. 385–429, 2007.

[6] F. Baader, "Terminological cycles in a description logic with existential restrictions," in *Proc. of 18th Intl Joint Conf. on Artificial Intelligence (IJCAI)*, 2003, pp. 325–330.

[7] A. Calì, G. Gottlob, and T. Lukasiewicz, "A general Datalog-based framework for tractable query answering over ontologies," in *Proc. of the 28th Symp. on Principles of Database Systems (PODS)*, 2009, pp. 77–86.

[8] ——, "Datalog$^{\pm}$: A unified approach to ontologies and integrity constraints," in *Proc. of the 12th Intl Conf. on Database Theory (ICDT)*, 2009, pp. 14–30.

[9] A. Calì, G. Gottlob, and A. Pieris, "Advanced processing for ontological queries," in *Proc. of the 36th Intl Conf. on Very Large Databases (VLDB)*, 2010, pp. 554–565.

[10] ——, "Query answering under non-guarded rules in Datalog$^{\pm}$," in *Proc. of the 4th Intl conf. on Web Reasoning and Rule Systems (RR)*, 2010, pp. 1–17.

[11] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, 1989.

[12] C. Beeri and M. Y. Vardi, "A proof procedure for data dependencies," *Journal of the ACM*, vol. 31, no. 4, pp. 718–741, 1984.

[13] M. Y. Vardi, "On the complexity of bounded-variable queries," in *Proc. of the 14th Symp. on Principles of Database Systems (PODS)*, 1995, pp. 266–276.

[14] A. Calì, G. Gottlob, and A. Pieris, "Query rewriting under non-guarded rules," in *Proc. of the 4th Alberto Mendelzon Intl Work. on Foundations of Data Management (AMW)*, 2010.

[15] A. Deutsch, L. Popa, and V. Tannen, "Query reformulation with constraints," *SIGMOD Record*, vol. 35, pp. 65–73, 2006.

[16] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle, "The ICS-FORTH RDFSuite: Managing voluminous RDF description bases," in *Proc. of the 2nd Intl Workshop on the Semantic Web (SemWeb)*, 2001, pp. 109–113.

[17] E. Chong, S. Das, G. Eadon, and J. Srinivasan, "An efficient SQL-based RDF querying scheme," in *Proc. of the 31th Intl Conf. on Very Large Data Bases (VLDB)*, 2005, pp. 1216–1227.

[18] G. Gottlob, N. Leone, and F. Scarcello, "Hypertree decompositions and tractable queries," in *In Proc. of the 18th Symp. on Principles of database systems (PODS)*, 1999, pp. 21–32.

[19] H. Pérez-Urbina, B. Motik, and I. Horrocks, "Efficient query answering for OWL 2," in *Proc. of the 8th Intl Semantic Web Conf. (ESWC)*, 2009, pp. 489–504.

[20] R. Rosati and A. Almatelli, "Improving query answering over DL-Lite ontologies," in *In Proc. of the 20th Intl Conf. on Principles of Knowledge Representation (KR)*, 2010.

[21] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," in *Proc. of the 9th ACM Symp. on Theory of Computing (STOC)*, 1977, pp. 77–90.

[22] A. Halevy, "Answering queries using views: A survey," *The VLDB Journal*, vol. 10, pp. 270–294, 2001.

[23] A. Deutsch and V. Tannen, "Mars: A system for publishing XML from mixed and redundant storage," in *In Proc. of the 29th Intl Conf. on Very large data bases (VLDB)*, 2003, pp. 201–212.

[24] C. Beeri and M. Y. Vardi, "The implication problem for data dependencies," in *Proc. of the 8th Colloquim on Automata, Languages and Programming (ICALP)*, 1981, pp. 73–85.

[25] A. Calı, G. Gottlob, and M. Kifer, "Taming the infinite chase: Query answering under expressive relational constraints," in *Proc. of the 11th Intl Joint Conf. on Principles of Knowledge Representation and Reasoning (KR)*, 2008, pp. 70–80.

[26] A. Deutsch, A. Nash, and J. Remmel, "The chase revisited," in *Proc. of the 27th Symp. on Principles of Database Systems (PODS)*, 2008, pp. 149–158.

[27] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Trans. on Database Systems*, vol. 4, no. 4, pp. 455–469, 1979.

[28] D. S. Johnson and A. C. Klug, "Testing containment of conjunctive queries under functional and inclusion dependencies," *Journal of Computer and System Sciences*, vol. 28, no. 1, pp. 167–189, 1984.

[29] R. Fagin, P. Kolaitis, R. Miller, and L. Popa, "Data exchange: Semantics and query answering," *Theoretical Computer Science*, vol. 336, no. 1, pp. 89–124, 2005.

[30] B. Marnette, "Generalized schema-mappings: From termination to tractability," in *In Proc. of the 28th Symp. on Principles of Database Systems (PODS)*, 2009, pp. 13–22.

[31] A. Calì, G. Gottlob, and A. Pieris, "Tractable query answering over conceptual schemata," in *Proc. of the 28th Intl Conf. on Conceptual Modeling (ER)*, 2009, pp. 175–190.

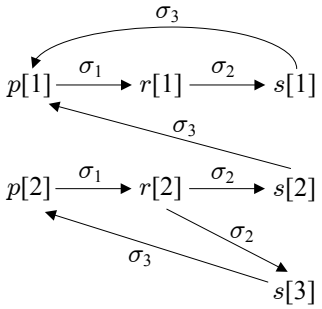
[32] ——, "Query answering under expressive Entity-Relationship schemata," in *Proc. of the 29th Intl Conf. on Conceptual Modeling (ER)*, 2010, pp. 347–361.

[33] A. K. Chandra and M. Y. Vardi, "The implication problem for functional and inclusion dependencies is undecidable," *SIAM Journal of Computing*, vol. 14, no. 3, pp. 671–677, 1985.

[34] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[35] A. Calì, D. Lembo, and R. Rosati, "On the decidability and complexity of query answering over inconsistent and incomplete databases," in *Proc. of the 22nd Symp. on Principles of Database Systems (PODS)*, 2003, pp. 260–271.

[36] R. de Virgilio, G. Orsi, L. Tanca, and R. Torlone, "Semantic data markets: a flexible environment for knowledge management," in *Proc. of 20th Intl Conf. on Information and Knowledge Management (CIKM)*, 2011, pp. 1559–1564.


σ_3
σ_1
σ_2
$p[1]$
$r[1]$
$s[1]$
σ_3
σ_1
σ_2
$p[2]$
$r[2]$
$s[2]$
σ_2
σ_3
$s[3]$